

Use of USB Port as a Host for USB Key

Targeted competences: Use of USB in order to connect an USB key to the microcontroller

Hardware: STM32F7 Nucleo board

Framework: STM32 CubeMX 5.5.0 and Keil μ vision V5.25.2.0

The aim of this document is to show how to use an USB port to connect a USB key to the microcontroller.

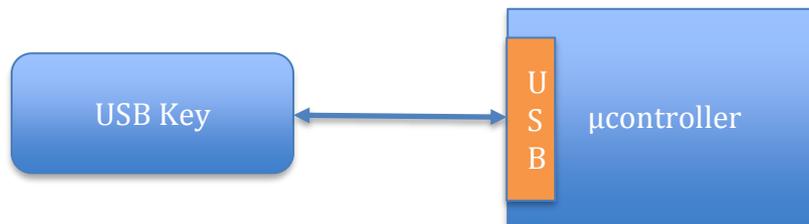


Figure 1: Global view of the system

We develop here a small code able to write a file on the usb key.

1. Microcontroller configuration

The first step is to configure the microcontroller. In our case, we use the NUCLEO-F767ZI platform based on a STM32F7 architecture. In order to configure this board we will use the CubeMx software. The first step after choosing the board is to name the project.

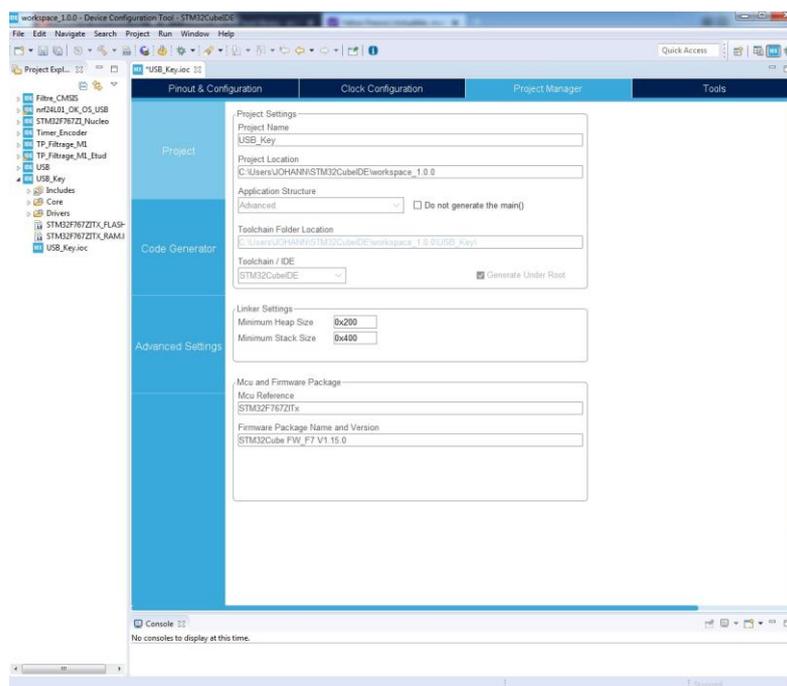


Figure 2: CubeIDE interface

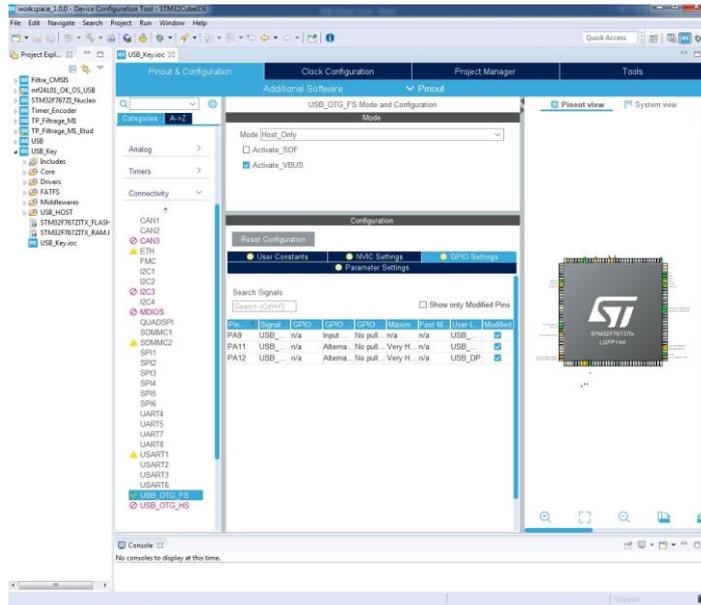


Figure 3: CubeMx interface to configure the STM32 Device

Here, I chose to use the USB_OTG_FS in host_only mode since we want to use an usb key as mass storage and no other peripheral that leads the need to use the usb port in device mode for instance. As we have to power the usb key, we need to set Activate_VBUS. Caution activate VBUS is not sufficient to power the usb key, we will have to write some code line to obtain the powering of the usb key.

The next step is to configure the usb in host mode. The figure below shows the configuration to realize.

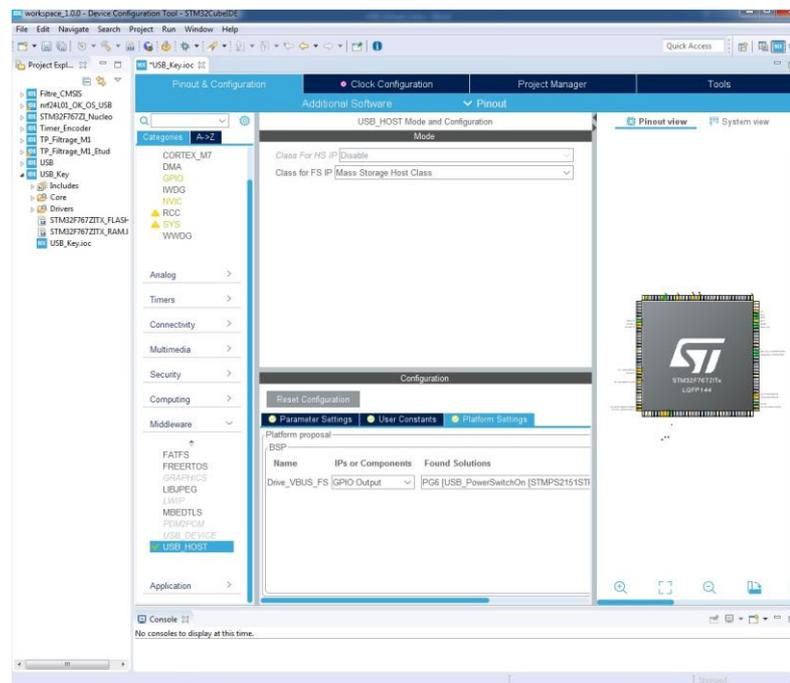


Figure 4: Host mode configuration

Here, we have to choose the class for the FS IP ; as we have chosen to use an usb key, we select the Mass Storage Host Class. The last step is the select the platform settings tab to choose a GPIO to drive the VBUS; we have to choose the PG6 pin. Finally, the last step is to select the FATFS module in order to generate a file system for the USB Disk. Here I left the default settings.

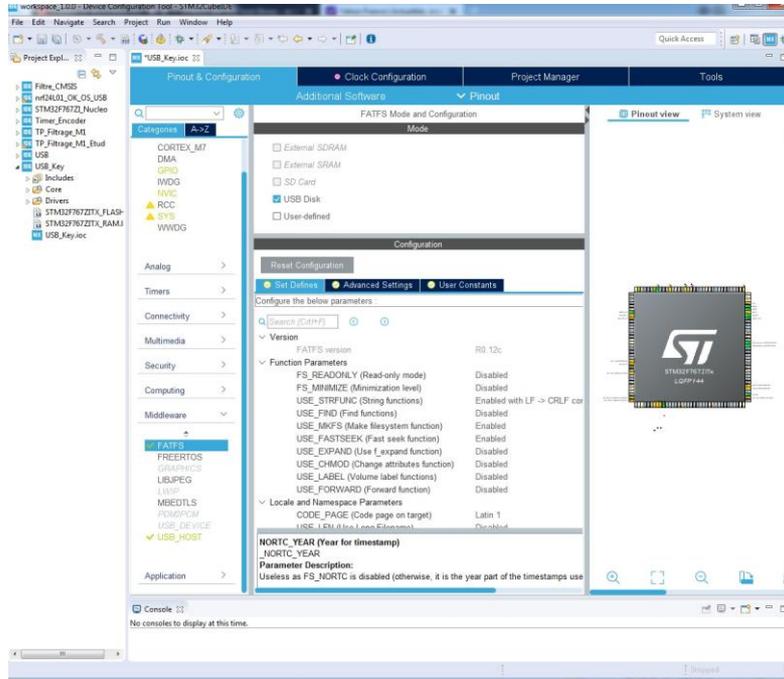


Figure 5: FATFS Configuration

Now we have to configure the clock for the whole board; to do this click on clock configuration item. With this configuration panel, we can choose the frequency of different μ controller components as CPU frequency, AHB, APB1 buses and so on. I decide to use the maximum frequency of the SYSCLK that is 216MHz. In this case, the frequency of the APB1 and APB2 timer clocks are respectively 108MHz and 216MHz (see Figure 6).

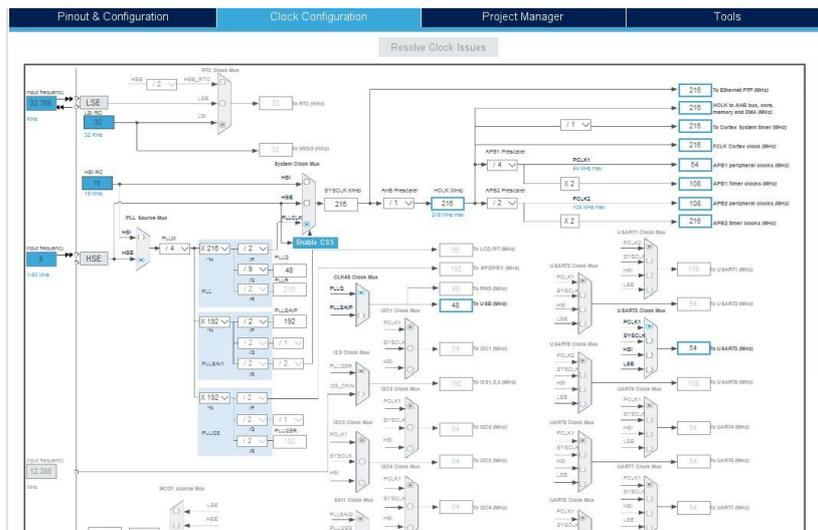


Figure 6: Clock configuration for the board

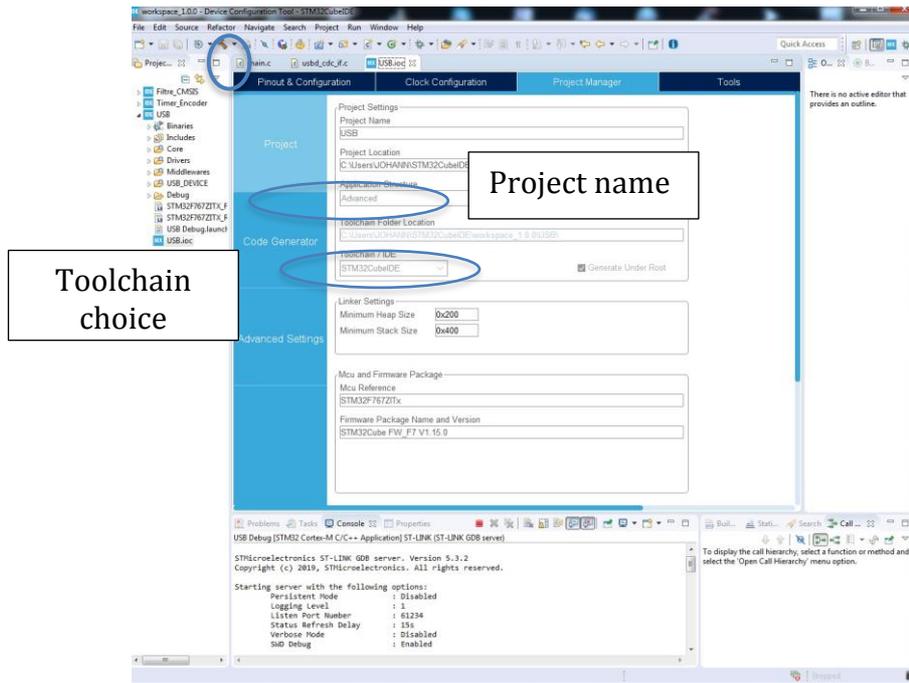


Figure 7: Project manager configuration

We have already given a name for our project and you can modify if you need the stack and heap parameters. Be careful if you use dynamic memory allocation indeed you should try to estimate the maximum memory size you need in order to choose the best heap size. If the heap size is too small, you will have some bugs during the program execution.

After clicking on Generate Code, CubeMx generates the application code and creates the project as shown in the figure below.

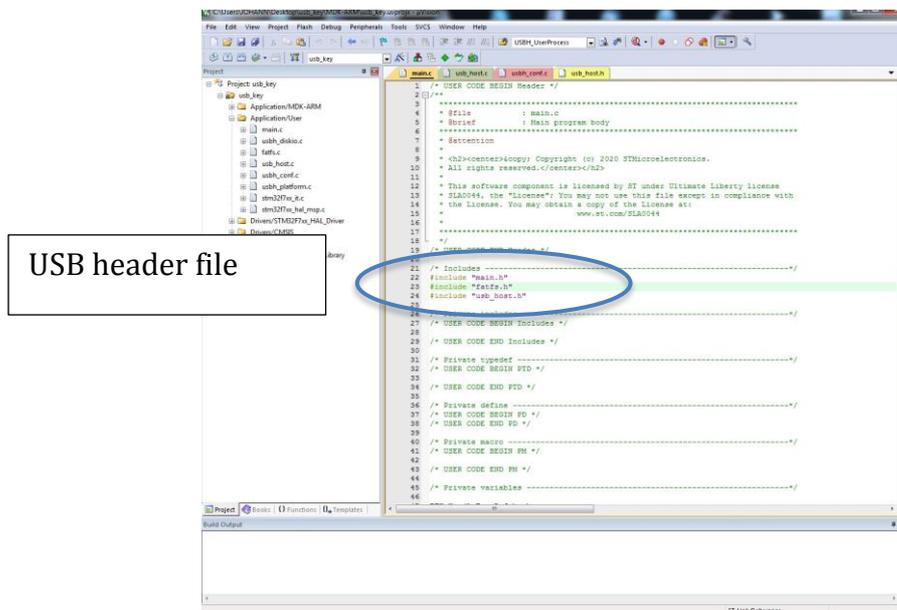


Figure 8: Code generation

In order to use the USB and FATFS APIs, we have to define some variables. I declared a file system object named USBDisk, a file object named FileToWrite, a variable

to store the path of the usb key and a variable to store the name of the file to write named NameOutputFile. Finally, I declared the Appli_state as an extern variable since this variable is defined in the usb_host.c file. I will use this variable in the main function to call the function Application when this variable is equal to Application_READY.

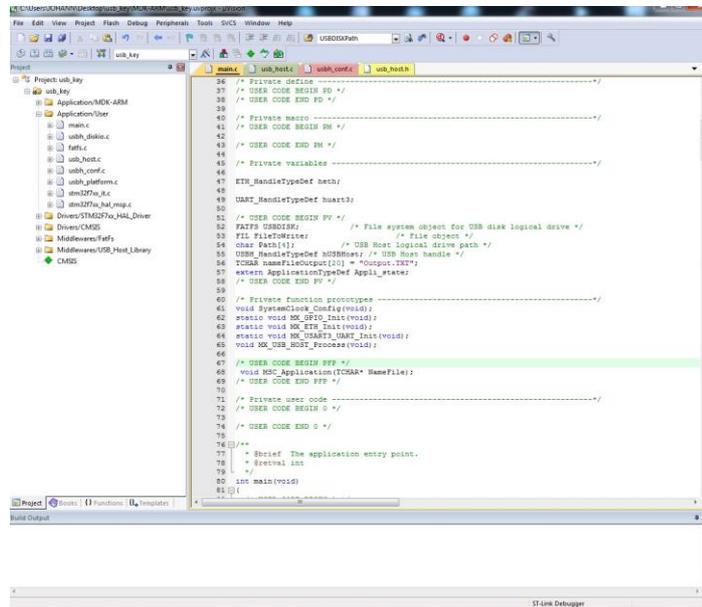


Figure 9: Variable declarations

In order to generate the VBUS signal to power the USB port, we have to modify the USBH_LL_DriverVBUS function since this function generated by the CubeMx tool is not functional.

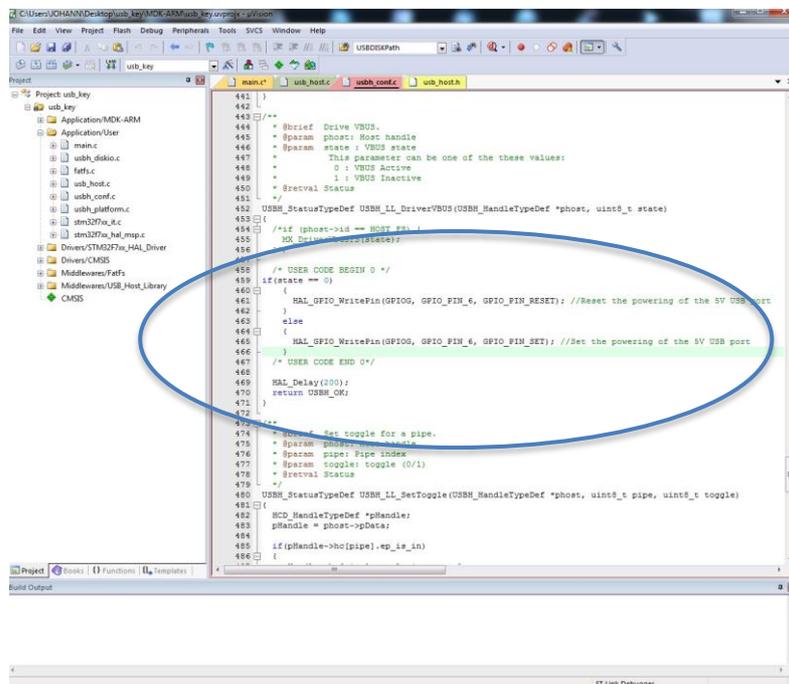


Figure 10: USBH_LL_Driver_DriverVBUS modification

```

81 int main(void)
82 {
83     /* USER CODE BEGIN 1 */
84     /* USER CODE END 1 */
85     /* Enable I-Cache-----*/
86     SCB_EnableICache();
87     /* Enable D-Cache-----*/
88     SCB_EnableDCache();
89
90     /* MCU Configuration-----*/
91     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
92     HAL_Init();
93
94     /* USER CODE BEGIN Init */
95     /* USER CODE END Init */
96
97     /* Configure the system clock */
98     SystemClock_Config();
99     /* USER CODE BEGIN SysInit */
100    /* USER CODE END SysInit */
101
102    /* Initialize all configured peripherals */
103    MX_GPIO_Init();
104    MX_ETH_Init();
105    MX_USART1_UART_Init();
106    MX_FATFS_Init();
107    MX_USB_HOST_Init();
108    /* USER CODE BEGIN 2 */
109    /* USER CODE END 2 */
110
111    /* Infinite loop */
112    /* USER CODE BEGIN WHILE */
113    while (1)
114    {
115        /* USER CODE END WHILE */
116        MX_USB_HOST_Process();
117
118        /* USER CODE BEGIN 3 */
119        if (Appl_state == APPLICATION_READY)
120        {
121            MSC_Application(nameFileOutput);
122        }
123        /* USER CODE END 3 */
124    }
125    ...

```

Figure 11: The main function code

In the While(1) part of the main.c I added some code in order to verify if the state machine of the usb core is in APPLICATION_READY state. In this state the usb port is ready to use so then I call the MSC_Application function with in parameter the name of the output file where I will write some text.

```

317 void MSC_Application(TCHAR* NameFile)
318 {
319     FRESULT res;
320     /* FatFs function common result code */
321     uint32_t byteswritten; /* File write counts */
322     uint8_t wsect[] = "this is a text for the USB tutorial"; /* File write buffer */
323
324     /* Register the file system object to the FatFs module */
325     if (!f_mount(&USBDISK, (TCHAR const*)Path, 0) != FR_OK)
326     {
327         /* FatFs Initialization Error */
328         Error_Handler();
329     }
330     else
331     {
332         /* Create and Open a new text file object with write access */
333         if (f_open(&FileToWrite, NameFile, FA_CREATE_ALWAYS | FA_WRITE) != FR_OK)
334         {
335             /* STM32.TXT file Open for write Error */
336             Error_Handler();
337         }
338         else
339         {
340             /* Write data to the text file */
341             res = f_write(&FileToWrite, wsect, sizeof(wsect), (void *)&byteswritten);
342             HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, GPIO_PIN_SET); // Switch on the led 1
343             HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Switch on the led 2
344             if (byteswritten == 0) || (res != FR_OK)
345             {
346                 /* File Write or EOF Error */
347                 Error_Handler();
348             }
349             else
350             {
351                 /* Close the open text file */
352                 f_close(&FileToWrite);
353             }
354         }
355     }
356
357     /* Unlink the USB disk I/O driver */
358     FATFS_UnLinkDriver(Path);
359 }
360

```

Figure 12: Code to write the text in the file in the USB key

At the end we have to compile the code in order to obtain the executable file for the STM32F7. When the compilation is done without error, we can transfer the code into the board.

A video showing the test of the code can be found here: <https://youtu.be/43Rkx9ilfQo>